Trabajo Práctico 3
Microarquitectura

Tecnología Digital II

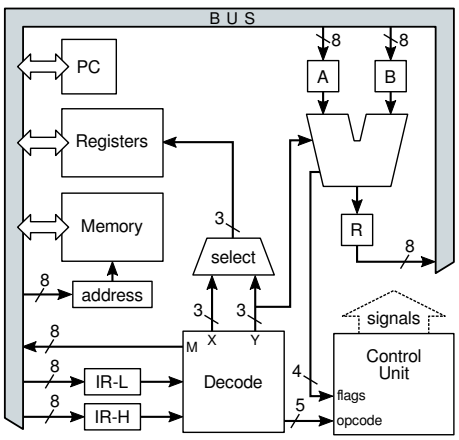
Introducción

El presente trabajo práctico consiste en analizar y extender una microarquitectura diseñada sobre el simulador *Logisim*. Se buscará codificar programas simples en ensamblador, modificar parte de la arquitectura y diseñar nuevas instrucciones.

El simulador se puede bajar desde la página `http://www.cburch.com/logisim/` o de los repositorios de Ubuntu. Requiere Java 1.5 o superior. Para ejecutarlo, ingresar en una consola:
`java -jar logisim.jar`.

> El trabajo práctico debe realizarse en grupos de tres personas. Tienen dos semanas para realizar la totalidad de los ejercicios y entregar un informe en formato digital con la solución de los ejercicios.
> **La fecha de entrega límite es el lunes 5/6 a las 23:59.**
> Se solicita no realizar consultas del trabajo práctico por los foros públicos. Limitar las preguntas al foro privado creado para tal fin.

Procesador `OrgaSmall`



- Arquitectura *von Neumann*, memoria de datos e instrucciones compartida.
- 8 registros de propósito general, `R0` a `R7`.
- 1 registro de propósito específico PC.
- Tamaño de palabra de 8 bits y de instrucciones 16 bits.
- Memoria direccionable a byte de tamaño 256 bytes.
- Bus de 8 bits.
- Diseño microprogramado.

Para poder descargar la Arquitectura OrgaSmall ir a: `https://github.com/fokerman/microOrgaSmall/`
La versión que utilizaremos se encuentra bajo el nombre `OrgaSmallWithStack`.

Ejercicios

1. **Introducción** - Leer la hoja de datos y responder:

   a) ¿Cuál es el tamaño de la memoria en cantidad de bytes?

   b) ¿Cuántas instruciones sin operandos se podrían agregar al formato de instrucción?

   c) ¿Qué tamaño tiene el `PC`?

   d) ¿Dónde se encuentra y qué tamaño tiene el `IR`?

   e) ¿Cual es el tamaño de la memoria de microinstrucciones? ¿Cuál es su unidad direccionable?

2. **Analizar** - Estudiar el funcionamiento de los circuitos indicados y responder las siguientes preguntas:

   a) `PC` (Contador de Programa): ¿Qué función cumple la señal `inc`?

   b) `ALU` (Unidad Aritmético Lógica): ¿Qué función comple la señal `opW`?

   c) `ControlUnit` (Unidad de control): ¿Cómo se resuelven los saltos condicionales? Describir detalladamente el mecanismo, incluyendo la forma en que interactuan las señales `jc_microOp`, `jz_microOp`, `jn_microOp` y `jo_microOp`, con los flags.

   d) `microOrgaSmall` (`DataPath`): ¿Para qué sirve la señal `DE_enOutImm`? ¿Qué parte del circuito indica cuál índice del registro a leer y escribir?

3. **Ensamblar y ejecutar** - Escribir el siguiente archivo, compilarlo y cargarlo en la memoria de la máquina:

```
start:
        SET R7, 0xFF
        SET R0, 0x01
        SET R1, 0x00
        SET R2, 0x10
        SET R3, 0x30

loop:
        CALL |R7|, shift
        SUB R2, R0
        CMP R1, R2
        JZ end
JMP loop
end:
JMP end

shift:
        PUSH |R7|, R2
        SHL R2, 1
        STR [R3], R2
        ADD R3, R0
        POP |R7|, R2
        RET |R7|
```

> Para ensamblar el archivo, nombrarlo como `ejemplo.asm` y ejecutar el siguiente comando:
> `python assembler.py ejemplo.asm`
> Este comando genera un archivo `.mem` que puede ser cargado en la memoria RAM de la máquina. Además, genera un archivo `.txt` con las instrucciones en ensamblador del programa y sus direcciones de memoria para facilitar la lectura del binario.

   a) Previamente a ejecutar el programa, describir con palabras el comportamiento esperado del mismo. No se debe explicar instrucción por instrucción, la idea es entender qué hace el programa y qué resultado genera.

   b) Identificar la dirección de memoria de cada una de las etiquetas del programa.

   c) Ejecutar e identificar cuántos ciclos de clock son necesarios para que el programa llegue a la instrucción `JMP halt`.

   d) ¿Cuántas microinstrucciones son necesarias para ejecutar la instrucción `ADD`? ¿Cuántas para la instrucción `JZ`? ¿Cuántas para la instrucción `JMP`?

   e) Describir detalladamente el funcionamiento de las instrucciones `PUSH`, `POP`, `CALL` y `RET`.

4. **Programar** - Escribir en ASM las siguientes funciones:

   a) Escribir la función `cantPares` que toma un array de enteros positivos en memoria y cuenta cuantos elementos pares tiene.

```
cantPares(*p,size)
        for f=0; f<size; f++
                if (p[f] es par) count++
        return count
```

b) Escribir otra función `modArray` que toma el mismo array del punto anterior y modifica sus valores, dividiendo por 4 y restando uno a los multiplos de 4 y multiplicando por 5 y restando uno al resto.

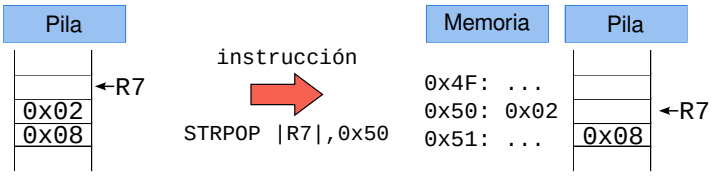
```
modArray(*p,size)
        for f=0; f<size; f++
                if (p[f] es mult4) p[f]=p[f]/4
                else p[f]=p[f]*5-1
```

Considerar en todos los casos que los parámetros llegan en `R0` y `R1`. El resultado debe ser guardado en `R4`. Considerar que ningún registro debe ser modificado, excepto el `R4`.
Para este ejercicio se proporciona un conjunto de archivos base donde pueden completar la implementación de sus funciones. Estos archivos pueden ser modificados para complementar su entrega con otros ejemplos de datos de entrada.

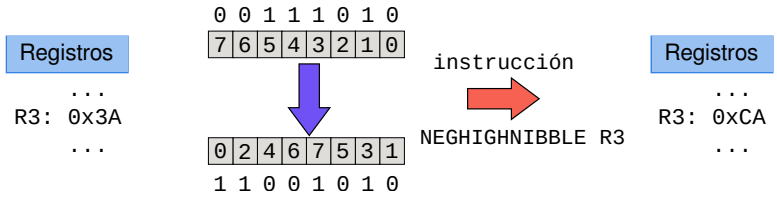
5. **Ampliando la máquina** - Agregar las siguientes nuevas instrucciones:

> Para generar un nuevo *set* de microinstrucciones, generar un archivo `.ops` y traducirlo a señales con el siguiente comando:
> `python buildMicroOps.py NombreDeArchivo.ops`
> Este generará un archivo `.mem` que puede ser cargado en la memoria ROM de la Unidad de Control.

a) Sin agregar circuitos nuevos, agregar la instrucción `STRPOP` que almacena en la dirección pasada como parámetro el número almacenado en la pila. Esta instrucción debe dejar la pila consistente, es decir, quitando el dato de la pila. Se recomienda utilizar como código de operación el 0x0E



b) Agregar la instrucción `NEGHIGHNIBBLE`, que hace el inverso de los 4 bits más significativos de un registro sin modificar los otros 4. El resultado se almacena en el mismo registro. Para implementar esta instrucción se debe modificar el circuito de la ALU para lo operación 14. Bajo esté código se debe agregar la operación de la ALU que realice el intercambio de bits. Se recomienda utiliza como código de operación el 0x0F



Nota: cada ítem debe ser presentado con un código de ejemplo que pruebe la funcionalidad agregada.